```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
  initialize the search tree using the initial state of problem
  loop do
    if there are no candidates for expansion then return failure
    choose a leaf node for expansion according to strategy
    if the node contains a goal state then return the corresponding solution
    else expand the node and add the resulting nodes to the search tree
  end
```

```
function GRAPH-SEARCH(problem, fringe) return a solution, or failure
  closed ← an empty set
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe)
    if GOAL-TEST(problem, STATE[node]) then return node
    if STATE[node] is not in closed then
      add STATE[node] to closed
      for child-node in EXPAND(STATE[node], problem) do
        fringe ← INSERT(child-node, fringe)
      end
  end
```

DFS, m level, b children of one node,
Time $O(b^m)$ Space $O(bm)$
Is it optinal? No, it finds the "leftmost" solution, regardless of depth or cost
BFS time $O(b^s)$ s is the depth of solution
Space $0(b^s)$, ans it is complete, it is optimal if costs are all 1
UCS = uniform cost search
Is guaranteed to return an optimal path
Expand a cheapest node first, akes time $O(b^{C^*/\varepsilon})$, If that solution costs C^* and arcs cost at least ε, then the "effective depth" is roughly C^*/ε, space is the same,
Processes all nodes with cost less than cheapest solution!
Search Heuristics
Admissible: 0<= h(n)<=h*(n)
h*(n) is the true cost to a nearest goal
Consistency: heuristic "arc" cost ≤ actual cost for each arc h(A) – h(C) ≤ cost(A to C)
Greedy Search-best first search
Expand a mode that you think is closest to a goal state
A* Search
Admissible heuristics never underestimate? False 不会高估 cost
Uniform-cost =g(n), greedy-cost= h(n)
A* Search orders by the sum: f(n) = g(n) + h(n), stop: when we de-queue a goal
The different between a* and ucs is that a* add the heuristics which will intend to lead the agent to the goal with a purpose
Any complete search algorithm must have exponential (or worse) time complexity. True! (why?)
BFS always returns the optimal cost path. False (in general)
Before expanding a node, check to make sure its state has never been expanded before
If not new, skip it, if new add to closed set
Important: store the closed set as a set, not a list
Constraint Search problem
Backtracking Search: Depth-first search with these two improvements is called *backtracking search* (not the best name)
Idea 1: One variable at a time
Idea 2: Check constraints as you go
Improvement: Arc consistency & forward checking

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
  return RECURSIVE-BACKTRACKING({ }, csp)
function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment given CONSTRAINTS[csp] then
      add {var = value} to assignment
      result ← RECURSIVE-BACKTRACKING(assignment, csp)
      if result ≠ failure then return result
      remove {var = value} from assignment
  return failure
```

```
function AC-3( csp) returns the CSP, possibly with reduced domains
  inputs: csp, a binary CSP with variables {X1, X2, ..., Xn}
  local variables: queue, a queue of arcs, initially all the arcs in csp

  while queue is not empty do
    (Xi, Xj) ← REMOVE-FIRST(queue)
    if REMOVE-INCONSISTENT-VALUES(Xi, Xj) then
      for each Xk in NEIGHBORS[Xi] do
        add (Xk, Xi) to queue
---------------------------------------------------------------
function REMOVE-INCONSISTENT-VALUES( Xi, Xj) returns true iff succeeds
  removed ← false
  for each x in DOMAIN[Xi] do
    if no value y in DOMAIN[Xj] allows (x,y) to satisfy the constraint Xi ↔ Xj
      then delete x from DOMAIN[Xi]; removed ← true
  return removed
```

Runtime: $O(n^2d^3)$, can be reduced to $O(n^2d^2)$
but detecting all possible future problems is NP-hard
ARC consistency- also called 2-consistency-a kind of binary consistency can affect any assigned variable in the graph that has a path to the assigned variable
Limitation:1, can have one solution left. 2, have multiply solution left. 3, none
Ordering: choose the MRV(Minimum(choose value) remaining values), Choose the variable with the fewest legal left values in its domain, also called most constrained variable, fall-fast ordering
(which variable should be assigned next)
Ordering: Least Constraining Value
(choose node)
Given a choice of variable, choose the *least constraining value*
(in what order should its values be tired)
Iterative Improvement & local search: try incorrect solution and modify it to be correct.
Algorithm: while not solved:
randomly select any conflicted variable
Value selection: min-conflicts heuristic:
Choose a value that violates the fewest constraint
I.e., hill climb with h(n) = total number of violated constraints
local search: improve a single option until you can't make it better (no fringe!)
1,start whatever 2, repeat, move to the best neighboring state 3, if no better neighbors than current, return and quit
def max-value(state): def min-value(state):
initialize v = -∞
for each successor of state:
v = max(v, value(successor)) ; v = min(v, value(successor))
return v ***def value(state):***
if the state is a terminal state: return the state's utility
if the next agent is MAX: return max-value(state)if the next agent is MIN: return min-value(state) time, space :DFS(without depth limited)
Alpha-Beta Implementation
α: MAX's best option on path to root
β: MIN's best option on path to root
def max-value(state, α, β):
initialize v = -∞
for each successor of state:
v = max(v, value(successor, α, β))
if v ≥ β return v
α = max(α, v)
return v
def min-value(state , α, β):
initialize v = +∞
for each successor of state:
v = min(v, value(successor, α, β))
if v ≤ α return v
β = min(β, v)
return v
Time complexity drops to $O(b^{m/2})$

Markov Decision Processes- ***MDPs*** are non-deterministic search problems
One difference between MDP and expectimax: rewards are smeared throughout the tree(exit-max) instead of at the end.(MDP)
The value (utility) of a state s:
V^*(s) = expected utility starting in s and acting optimally
The value (utility) of a q-state (s,a):
Q^*(s,a) = expected utility starting out having taken action a from state s and (thereafter) acting optimally (bellman equation)

$$V^*(s) = \max_a Q^*(s,a))$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$$

$$V^*(s) = \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$$

T 是概率 V like MAX and Q like EXP. But now incorporates rewards and discounts. transition function
Optimal action: action with highest Q value

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V_k(s')\right]$$

Complexity of each iteration: $O(S^2A)$
How do we know the V_k vectors are going to converge?
Case 2: If the discount is less than 1
So as k increases, the values converge

$$V^\pi(s) = \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V^\pi(s')]$$

Utilities for a Fixed Policy
calculate the V's for a fixed policy π(Policy Evaluation)

$$V^\pi_{k+1}(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V^\pi_k(s')]$$

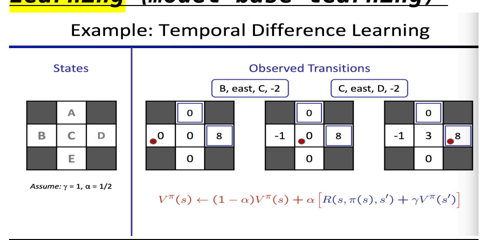
efficiency: $O(S^2)$ per iteration
problem:1, slow 2, the max at each state rarely changes 3, the policy often converges long before the values
policy iteration: Evaluation

$$V^{\pi_i}_{k+1}(s) \leftarrow \sum_{s'} T(s,\pi_i(s),s')\left[R(s,\pi_i(s),s') + \gamma V^{\pi_i}_k(s')\right]$$

improvement: For fixed values, get a better policy using policy extraction

$$\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^{\pi_i}(s')\right]$$

Example: Temporal Difference Learning (model base learning)



Value learning: watch some agent's performance under a policy and see how well it does
Q-learning: watch yourself perform, possibly suboptimally, and still derive optimal policies/values/etc.

$$V^\pi(s) \leftarrow (1-\alpha)V^\pi(s) + \alpha\left[R(s,\pi(s),s') + \gamma V^\pi(s')\right]$$

if we want to turn values into a (new) policy, we're sunk:

$$\pi(s) = \arg\max_a Q(s,a)$$

$Q(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V(s') \right]$

so keep a running averageß

$Q(s,a) \leftarrow (1-\alpha) Q(s,a) + (\alpha) \left[r + \gamma \max_{a'} Q(s',a') \right]$

approximate Q-learning

$\text{difference} = \left[r + \gamma \max_{a'} Q(s',a') \right] - Q(s,a)$

$Q(s,a) \leftarrow Q(s,a) + \alpha \, [\text{difference}]$

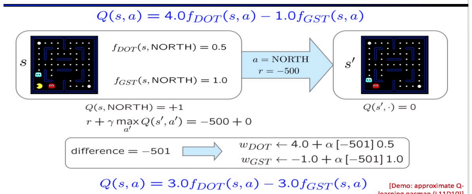
$w_i \leftarrow w_i + \alpha \, [\text{difference}] \, f_i(s,a)$

example:

$Q(s,a) = 4.0 f_{DOT}(s,a) - 1.0 f_{GST}(s,a)$

wdot = 4, wgst = −1

Example: Q-Pacman



conditional P: P(a|b)= P(a,b)/P(b)

product rule: $P(y)P(x|y) = P(x,y)$

$P(W|T=c)$ w sun; p =0.4

w rain: p = 0.6

Inference by Enumeration: Worst-case time complexity $O(d^n)$, Space complexity $O(d^n)$ to store the joint distribution

Chain rule:

$P(x_1, x_2, x_3) = P(x_1) P(x_2|x_1) P(x_3|x_1, x_2)$

Baye's rule: $P(x|y) = \frac{P(y|x)}{P(y)} P(x)$ =P(x, y)

P(y)= Σ(m)P(y,m)=P(+m)P(y|m)+P(-m)P(y|-m)

$= \sum_{x_{t-1}} P(x_t \mid x_{t-1}) P(x_{t-1})$

Sum =1

Using baye's rule: P(G) = 0.11 for all is uniform

P(r=yellow| ghost at(1,1))= 0.1

So P(g|r)= P(g)P(r|g)=0.1*0.11=0.011

X is conditionally independent of Y given Z:

$\forall x,y,z : P(x,y|z) = P(x|z)P(y|z)$

$\forall x,y,z : P(x|z,y) = P(x|z)$

Joint Distribution of a Markov Model

核心：now only related to past, future only related to now

$P(X_1, X_2, X_3, X_4) = P(X_1) P(X_2|X_1) P(X_3|X_2) P(X_4|X_3)$

$X_3 \perp\!\!\!\perp X_1 \mid X_2 \quad X_4 \perp\!\!\!\perp X_1, X_2 \mid X_3$

Stationary Distributions:最终 convegrent：

$P_\infty(X) = P_{\infty+1}(X) = \sum_x P(X|x) P_\infty(x)$

HMM: evidence 're only conditionally independent given the hidden state

Quiz: does this mean that evidence variables are guaranteed to be independent?

No, they tend to correlated by the hidden state]

Joint distribution of a HMM:

$P(X_1, E_1, X_2, E_2, X_3, E_3) = P(X_1) P(E_1|X_1) P(X_2|X_1) P(E_2|X_2) P(X_3|X_2) P(E_3|X_3)$

$P(X_1, E_1, \ldots, X_T, E_T) = P(X_1) P(E_1|X_1) \prod_{t=2}^{T} P(X_t|X_{t-1}) P(E_t|X_t)$

inference: base case:

$P(X_1|e_1)$

$P(x_1|e_1) = P(x_1, e_1)/P(e_1)$

$\propto_{X_1} P(x_1, e_1)$

$= P(x_1) P(e_1|x_1)$

if we have e_1 to e_t and the $B(X_t)=P(X_t|e_{1:t})$, we reason about : P(Xt+1|e1:t)= $= \sum_{x_t} P(X_{t+1}, x_t | e_{1:t})$

$= \sum_{x_t} P(X_{t+1}|x_t) P(x_t|e_{1:t})$

or we can say:

$B'(X_{t+1}) = \sum_{x_t} P(X'|x_t) B(x_t)$

this is the guess part

the following is the observation part:

we get the believe for Xt+1:

$B'(X_{t+1}) = P(X_{t+1}|e_{1:t})$

then the evidence comes in

$P(X_{t+1}|e_{1:t+1}) =$

$P(X_{t+1}, e_{t+1}|e_{1:t}) / P(e_{t+1}|e_{1:t})$

$\propto_{X_{t+1}} P(X_{t+1}, e_{t+1}|e_{1:t})$

$= P(e_{t+1}|e_{1:t}, X_{t+1}) P(X_{t+1}|e_{1:t})$

$= P(e_{t+1}|X_{t+1}) P(X_{t+1}|e_{1:t})$

Unlike passage of time, we have to renormalize

Basic idea: beliefs "reweighted" by likelihood of evidence, and compactly:

$B(X_{t+1}) \propto_{X_{t+1}} P(e_{t+1}|X_{t+1}) B'(X_{t+1})$

the forward algorithm

we want to know $B_t(X) = P(X_t|e_{1:t})$

we can derive the following updates:

$P(x_t|e_{1:t}) \propto_X P(x_t, e_{1:t})$

$= \sum_{x_{t-1}} P(x_{t-1}, x_t, e_{1:t})$

$= \sum_{x_{t-1}} P(x_{t-1}, e_{1:t-1}) P(x_t|x_{t-1}) P(e_t|x_t)$

$= P(e_t|x_t) \sum_{x_{t-1}} P(x_t|x_{t-1}) P(x_{t-1}, e_{1:t-1})$

SUMMAY:

We reason that $P(x_t|e_{1:t-1}) = \sum_{x_{t-1}} P(x_{t-1}|e_{1:t-1}) \cdot P(x_t|x_{t-1})$

Get evidence: we get: (update)

$P(x_t|e_{1:t}) \propto_X P(x_t|e_{1:t-1}) \cdot P(e_t|x_t)$

Particle filter:

Step 1: base on the evidence-> guess what will happen

$x' = \text{sample}(P(X'|x))$

Step 2: observe: $w(x) = P(e|x)$

$B(X) \propto P(e|X) B'(X)$ As before, the probabilities don't sum to one, since all have been down weighted

Step 3: resample: normalize

HMMS: most likely explanation: forward algorithm (Sum)

$\arg\max_{x_{1:t}} P(x_{1:t}|e_{1:t})$ $f_t[x_t] = P(x_t, e_{1:t})$

Viterbi Algorithm(max): $m_t[x_t] = \max_{x_{1:t-1}} P(x_{1:t-1}, x_t, e_{1:t})$

$= P(e_t|x_t) \max_{x_{t-1}} P(x_t|x_{t-1}) m_{t-1}[x_{t-1}]$

which state sequence $X_{1:T}$ is most likely given the evidence e 1:t?

$x^*_{1:T} = \arg\max_{x_{1:T}} P(x_{1:T}|e_{1:T}) = \arg\max_{x_{1:T}} P(x_{1:T}, e_{1:T})$

Bayes Net: to see what probability a BN gives to a full assignment：假设 x 只与 parent（x）有关

$P(x_1, x_2, \ldots x_n) = \prod_{i=1}^{n} P(x_i|\mathit{parents}(X_i))$ same :

$P(x_i|x_1, \ldots x_{i-1}) = P(x_i|\mathit{parents}(X_i))$

How big is an N-node net if nodes have up to k parents?

$O(N * 2^{k+1})$

General question: in a given BN, are two variables independent (given evidence)?

X->Y->Z, Guaranteed X independent of Z ? *No!*

Guaranteed X independent of Z given Y? yes!

Y->X, Y->Z. Guaranteed X independent of Z ? *No!*

Guaranteed X independent of Z given Y? yes!

Evidence along the chain "blocks" the influence

X->Z, Y->Z. Are X and Y independent? Yes

Are X and Y independent given Z? NO: seeing traffic puts the rain and the ballgame in competition as explanation.

Active means no independent (dependent)

A trail X1–...–Xn is active if : it has no v-structures Xi–1→Xi←Xi+1

A trial X1–...–Xn is active given Z if :

1, for any v-structure Xi–1→Xi←Xi+1we have that Xi or one of its descendants∈Z 2, no other Xi is in Z

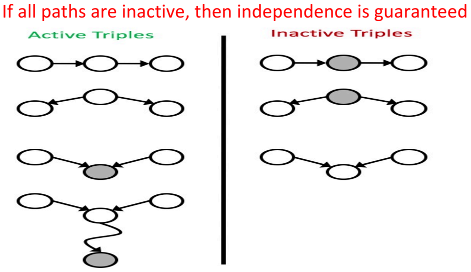
如果两个节点间没有 Active trial，则这两个节点是 d-separated 的 也就是独立的！ Independence

$X_i \perp\!\!\!\perp X_j | \{X_{k_1}, \ldots, X_{k_n}\}$? Given means Block (DARKED)

Check all (undirected!) paths between Xi and Xj

If one or more active path, then independence not guaranteed

If all paths are inactive, then independence is guaranteed



Decision Networks: **choose the action which maximizes the expected utility given the evidence**

= EU(ACTION) =

$\text{EU}(\text{leave}) = \sum_w P(w) U(\text{leave}, w)$

MEU(X)= max(EU for all)在当你获得新的 evidence 之后， you update you believe (probability), but the value of Utility doesn't change! Evidence kinds of forecast: P(rain |forecast = bad) = 0.66, P(sun | forecast = bad) = 0.34, recalculate the EU, and MEU

Value of information kinds of evidence =

MEU (after we get the information)- MEU(before we get the information)

$\text{VPI}(E'|e) = \left(\sum_{e'} P(e'|e) \text{MEU}(e, e') \right) - \text{MEU}(e)$

$\forall E', e : \text{VPI}(E'|e) \geq 0$

1, nonadditive:

$\text{VPI}(E_j, E_k|e) \neq \text{VPI}(E_j|e) + \text{VPI}(E_k|e)$

$\text{VPI}(E_j, E_k|e) = \text{VPI}(E_j|e) + \text{VPI}(E_k|e, E_j)$

$= \text{VPI}(E_k|e) + \text{VPI}(E_j|e, E_k)$

If Parents(U) ∏ Z | CurrentEvidence

Then VPI(Z | CurrentEvidence) = 0